nature machine intelligence

Supplementary information

https://doi.org/10.1038/s42256-025-01083-x

Towards generalizable and interpretable three-dimensional tracking with inverse neural rendering

In the format provided by the authors and unedited

Contents

Supplementary Code 1: Code Availability 2
Supplementary Video 1: Experimental Assessment for Dynamic Scenes 2
Supplementary Note 1: Additional Experimental Results 3
Supplementary Note 2: Additional Ablation Results 6
Supplementary Note 3: Interpretability of Failure Cases 9
Supplementary Note 4: Computational Cost 12
Supplementary Note 5: Differentiable Rendering Pipeline 14
Supplementary Note 6: Detailed Optimization Objectives 15
Supplementary Note 7: Detailed Optimization Schedule 18
Supplementary Note 8: Additional Tracking Details 20
Supplementary Note 9: Generative Object Model 24
Supplementary Note 10: Fair Comparison to End-to-end Approaches 26
Supplementary Note 11: Multi-Class and Human Generator Extension 27
Supplementary Note 12: Practical Impact of Interpretable INR-based Tracking 29
Supplementary Note 13: Validation of Retrieved Object Properties and Downstream Applications 30

In this supplementary document, we provide additional information, discussion, experiments, and results supporting the findings in the primary manuscript. To this end, we list training and architecture details, further ablation experiments, and additional comparisons and analysis. Code 1 provides access to the repository with the Supplementary Code. Video 1 showcases the tracking performance of our proposed approach on diverse scenes. In Note 1, we present further qualitative results for the 3D Multi-Object Tracking task on the nuScenes [1] and Waymo [2], including visualizations of rendered objects as well as additional details of the datasets we evaluate on. In Note 2, we present further ablations to validate our choices of training and optimization parameters used in our method. This is followed by a detailed study of the failure cases of our method via the visualizations provided by the rendered objects by our method in Note 3. In Note 4, we provide a breakdown of the computational cost of our inverse rendered optimization. Note 5 provides further details on the differentiable rendering pipeline at the heart of our method. Note 6 provides detailed descriptions of the optimization objective, followed by the optimization schedule used in inverse rendering steps in Note 7. In Note 8, we provide further details on the prediction, latent matching, kinematic state, and embedding update steps of our method, as well as tracking heuristics used. Although our method is not specific to any particular generative object prior, we describe the generative object model used in our evaluations in Note 9. In Note 10, we compare our method's generalization ability by providing a fair comparison to trained approaches via qualitative visualizations on an unseen dataset [2]. Note 11 evaluates the generalization of our approach to additional object classes aside from vehicles, the primary evaluation class. In Note 12, we provide further discussion on the practical impact of our method. Finally, in Note 13, we analyze the quality of retrieved object properties and discuss additional downstream applications.

Supplementary Code 1: Code Availability

We provide supplemental code to implement and run the multi-object tracking presented method. Following the steps in the README produces qualitative result videos for all validation scenes in nuScenes Mini. The Supplementary Code can be accessed under the following link https://github.com/princeton-computational-imaging/INRTracker.

Supplementary Video 1: Experimental Assessment for Dynamic Scenes

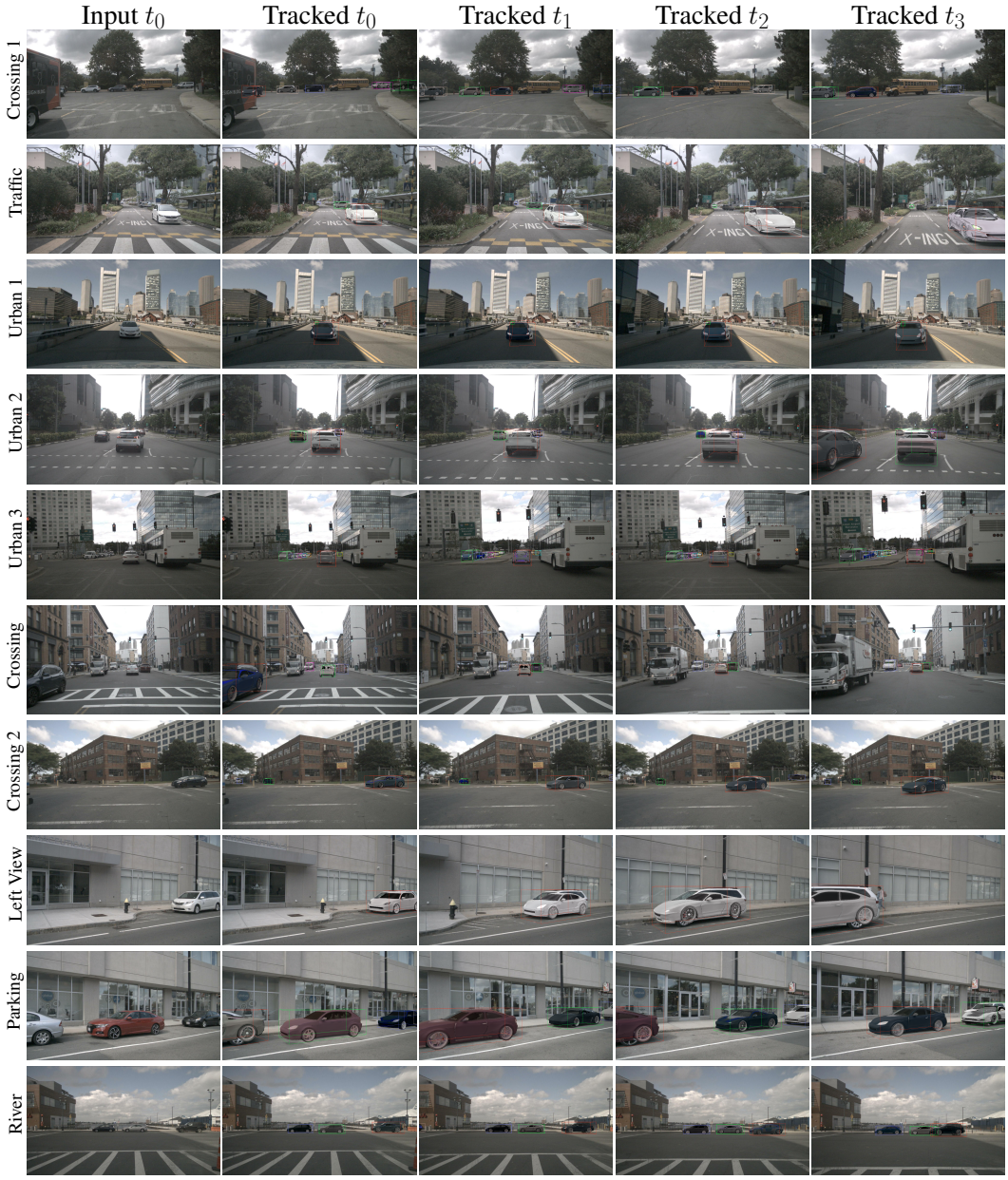
The Supplementary Video demonstrates the performance of our proposed INR-based tracking method on a sample of diverse scenes from the nuScenes [1] dataset and the Waymo Open Dataset [2]. We overlay the observed image with the rendered objects through alpha blending with a weight of 0.4. Object renderings are defined by the averaged latent embeddings $\mathbf{z}_{k,EMA}$ and the tracked object state $\mathbf{y}_k$.

Supplementary Note 1: Additional Experimental Results

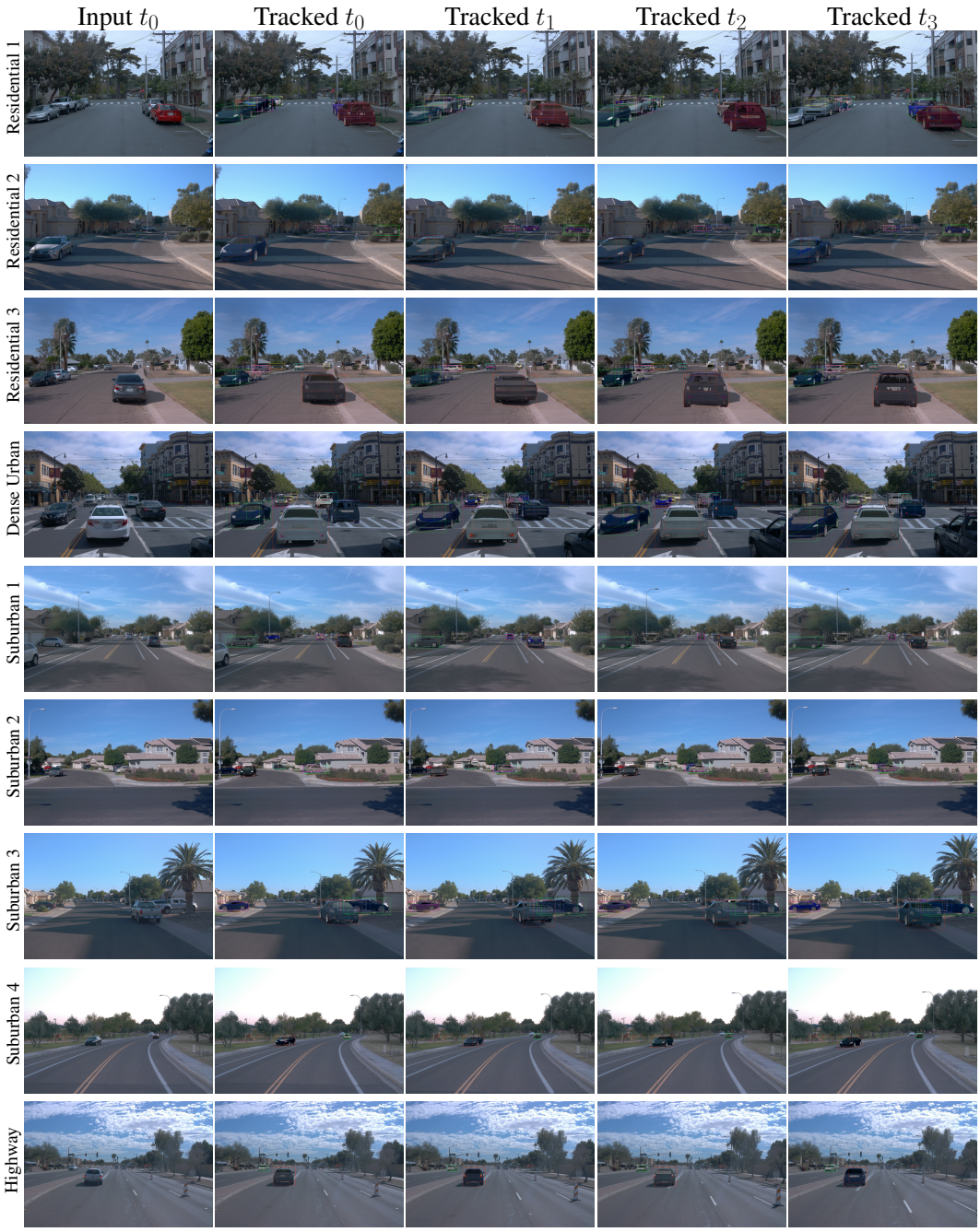
Next, we provide additional tracking results on the real-world datasets on which we test our method. While this section provides additional qualitative analysis of our proposed method's performance, see the evaluation section in the main paper for a detailed quantitative evaluation. Supplementary Figure 1 reports additional visualizations of the rendered RGB objects obtained by our proposed method on the nuScenes [1] dataset. Supplementary Figure 2 reports the same visualizations on the Waymo [2] dataset.

Additional Results on nuScenes Although the nuScenes [1] dataset consists of sensor data from 6 cameras, 5 radars, and 1 lidar, we tackle monocular camera-based 3D object tracking in this work. As such, we only use the data collected from the 6 cameras. The dataset comprises 1000 scenes, with each scene being 20s long. The test set contains 150 scenes. Each of these scenes is selected to be *interesting*, which include scenes with high traffic density (e.g., intersections, construction sites), rare classes (e.g. ambulances, animals), potentially dangerous traffic situations (e.g., jaywalkers, incorrect behavior), maneuvers (e.g., lane change, turning, stopping) and situations that may be difficult for an Autonomous Vehicle. Additional results of our method on the nuScenes dataset are listed in Fig. 1. We note that the colors of the cars are matched quite accurately. Moreover, the shapes of the cars get reconstructed as well. In turn, we can see that the tracking quality is high as visualized by bounding boxes. Note that the color of bounding boxes marks the same instance in consecutive frames.

Additional Results on Waymo Open Dataset The Waymo Open Dataset [2] consists of 1150 scenes that each span 20s. Again, since we tackle monocular tracking, we only use the data from the 5 camera sensors. The dataset was collected by driving in Phoenix AZ, Mountain View CA, and San Francisco CA across daytime, nighttime, and dawn lighting conditions. Additional results of our method on the Waymo Open dataset are given in Figure 2. We find that our method generalizes to this dataset. The colors of the cars are matched quite accurately, as shown in Figure 2. Moreover, the shapes of the cars get reconstructed as well. As such, again, tracking quality is high as visualized by bounding boxes. Note that the color of bounding boxes marks the same instance in consecutive frames.



Supplementary Figure 1: Additional visualizations on nuScenes [1]. From left to right, we show (i) observed images from diverse scenes at timestep $k = 0$; (ii) an overlay of the optimized generated object and its 3D bounding boxes at timestep $k = 0, 1, 2$ and 3. The color of the bounding boxes for each object corresponds to the predicted tracklet ID. We see that our method can accurately reconstruct objects in diverse scenarios.



Supplementary Figure 2: Additional visualizations on Waymo [2]. From left to right, we show (i) observed images from diverse scenes at timestep $k = 0$; (ii) an overlay of the optimized generated object and its 3D bounding boxes at timestep $k = 0, 1, 2$ and 3. The color of the bounding boxes for each object corresponds to the predicted tracklet ID. We see that our method can accurately match and track tracklets in diverse scenarios in the Waymo dataset as well, confirming that the method is dataset-agnostic.

Supplementary Note 2: Additional Ablation Results

We provide additional ablation results that further validate our choice of training and optimization parameters in our proposed method. Supplementary Table 1 shows the results of our hyperparameter search for each matching weight and the detection confidence threshold as denoted in Supplementary Note 6. Supplementary Table 2 shows the quantitative results from our ablation study of the optimization scheduler. Supplementary Figure 3 qualitatively compares the effect of our optimization schedule against using no schedule by showing the renderings of the optimized generations for both cases.

Matching Ablations. Adopting the same setting as AB3DMOT [3], we performed a hyperparameter search for each matching weight and the detection confidence threshold, as denoted in Sec. 4. Our full method outperforms AB3DMOT, see Table 1, conducted on the validation split. Our best setting outperforms AB3DMOT, the only other method not trained on the dataset, by 3.9% AMOTA on the nuScenes test split.

Optimization Schedule and Loss Ablations. As ablation experiments, we analyze the optimization schedule, the INR loss function components, and the weights of the tracker, applying them to a subset of scenes from the nuScenes validation set. We deliberately select this smaller validation set due to its increased difficulty. Tab. 2 lists the quantitative results from our ablation study of the optimization scheduler. Our findings reveal a crucial insight: the strength of our method lies not in isolated loss components but in their synergistic integration. Specifically, the amalgamation of pixel-wise, perceptual, and embedding terms significantly enhances AMOTA, MOTA, and Recall metrics.

Moreover, the absence of an optimization schedule led to less robust matching as quantitative and qualitative results in Tab. 3 reveal. However, the core efficacy of our tracking method remained intact as indicated in the last row of Tab. 2. This nuanced understanding underscores the importance of component interplay in our approach.

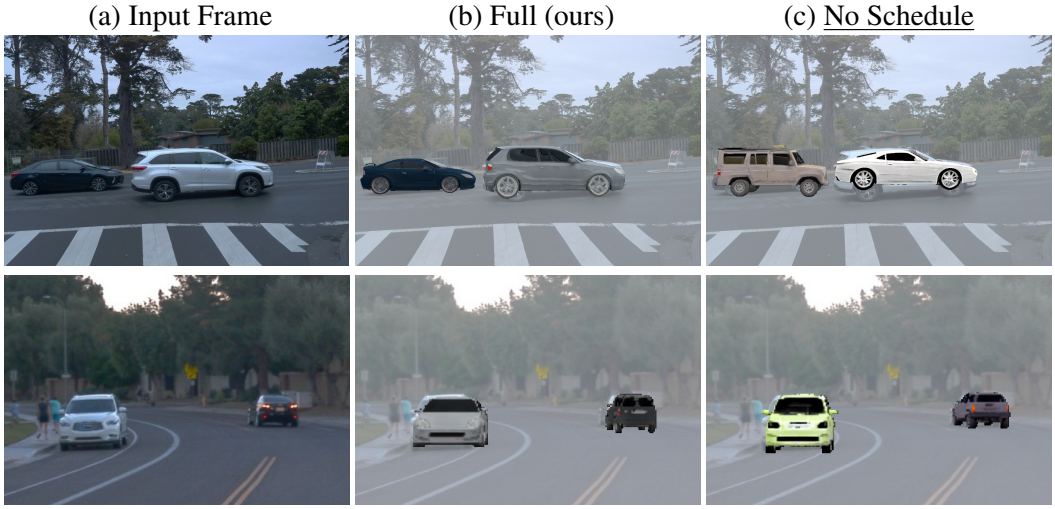
Supplementary Table 1: Tracking Matching and Detection Confidence. Parameters were optimized on the nuScenes [1] validation set. On the test split our best setting for w_{iou}, w_{center}, w_{embbed}, τ_{det} surpasses the performance of AB3DMOT [3], the only baseline not trained on the dataset.

Method (split)	**AMOTA** ↑	**Recall** ↑	**MOTA** ↑
AB3DMOT (CP, test)	0.387	0.506	0.284
Best Hyper-param. (CP, test)	**0.413**	**0.536**	**0.321**
$w_{iou} = 1.4$ (val)	0.403	0.540	0.322
$w_{center} = 0.9$ (val)	0.417	0.514	0.332
$w_{embedd} = 0.4$ (val)	0.418	0.558	0.332
$\tau_{det} = 0.4$ (val)	0.397	0.567	0.326

Supplementary Table 2: Optimization Schedule and Loss Components. Ablations were run on a small subset of the nuScenes [1] validation set. $\mathcal{L}_{RGB}$ fails due to the optimizer fitting objects to the background instead, increasing the size of each object resulting in out of memory.

Ablation Study of Loss Components.

Method	**AMOTA** ↑	**Recall** ↑	**MOTA**↑
$\mathcal{L}_{IR}$ & $\mathcal{L}_{embed}$	**0.112**	**0.264**	**0.113**
$\mathcal{L}_{IR}$	0.103	0.236	0.112
$\mathcal{L}_{perceptual}$	0.100	0.251	0.101
$\mathcal{L}_{RGB}$	N/A	N/A	N/A
No Schedule	0.102	0.224	0.110



Supplementary Figure 3: Effect of Optimization Schedule. (a) observed image, (b) optimized generations using the proposed schedule in Tab. 3 in Supplementary Note 7, (c) optimized generations using no schedule. This supports the quantitative results given in Tab. 2.

Supplementary Note 3: Interpretability of Failure Cases

Interpretability of Failure Cases. In this section, we provide additional analysis of the failure cases of our method. Our analysis via the RGB renderings produced by the object whose latent space we optimize over showcases the interpretable nature of our approach, providing a key advantage over prior works. Fig. 4 shows instances of failure cases of our method where the reconstructed object differs significantly from the observed object, showing that our approach allows us to identify possible improvements and understand why our model fails.

Visualizing the reconstructed objects allows us to reason about failure cases. For example, in scene **(e)** in Fig. 4, we see that the initial object significantly overlaps with the background asphalt in the shadow region, resulting in the optimization to converge towards a darker gray object appearance erroneously. Thus, our method allows us to reason about failure cases and identify ways our representation model can be modified to rectify such failures. For example, a generative object model with an additional component that can model different lighting conditions to account for shadows might allow us to identify and reconstruct cars in varied lighting conditions, including shadows. This can guide future work for perception tasks through inverse rendering.

Next, we analyze common failure cases using the pre-trained generator as an object prior to the presented tracking method. The visualizations allow us to assess cases where this pipeline fails to track objects. Common failure cases we observed are listed below, with visualizations of such failure cases shown in Figure 4. We describe the cases corresponding to the rows (a-d, see figure labels) as follows:

(a) The apparent darker color of the car due to **shadows** often causes the predicted object color to be darker than the color a human would perceive the car as. In the presented case on the right, the white car is completely occupied by the shadow of the neighboring truck. While the human visual system perceives the car's color as white, the numerical RGB value in the image is closer to gray/black. This causes the predicted embedding corresponding to the object's texture to represent the darker color. This might cause the tracking to fail due to incorrect matching of corresponding objects in consecutive frames with and without shadows.

(b) Extreme **reflections** on the car due to the lighting conditions cause the model to try to model the RGB color of the reflection by erroneously modifying the texture of the generated object. Here, clouds in the sky are reflected as white spots on the hood and windshield of the red car, causing reflection and influencing the generated texture as white spots. Future work that includes explicit models of BRDF would be beneficial in mitigating this class of failure modes.

(c) In addition to visualization and interpretation of the object prior more traditional aspects of the perception pipeline, such as ID-switches through **occlusion** in multi-object tracking scenarios can be observed. Here, the first object in the background (green box, t_0) is misidentified as the second object (green box, t_1). Such visualization allows further reasoning about the full pipeline.

(d) Camera **obstructions** and extreme local **lighting**, such as in raindrops, lens flares, and bright lights, cause our method to erroneously predict the shape and texture of many cars to match the perceived shape of the car, causing matching to fail.

(e) Inaccuracies in the predicted pose cause the predicted car in front to not perfectly align with the observed car patch, causing an overlap between the predicted car and the immediate surroundings in the observed image, here the asphalt. Since only information on the detection is available, the color of the optimized object tends to be predicted gray (since the road is gray, and so the optimized texture embedding is closer to gray).

Our generative prior does not model specular textures. Instead, it is restricted to diffuse reflectance. As such, it tends to reconstruct darker or lighter textures compensating for shadows from the environment and reflections of the sky. Modeling environmental lighting, complex material properties, and shadows may lead to a complex and less robust light simulation. Moreover, it will be restricted by the data available to train a generative prior model. Nevertheless, this is an exciting direction for future work.

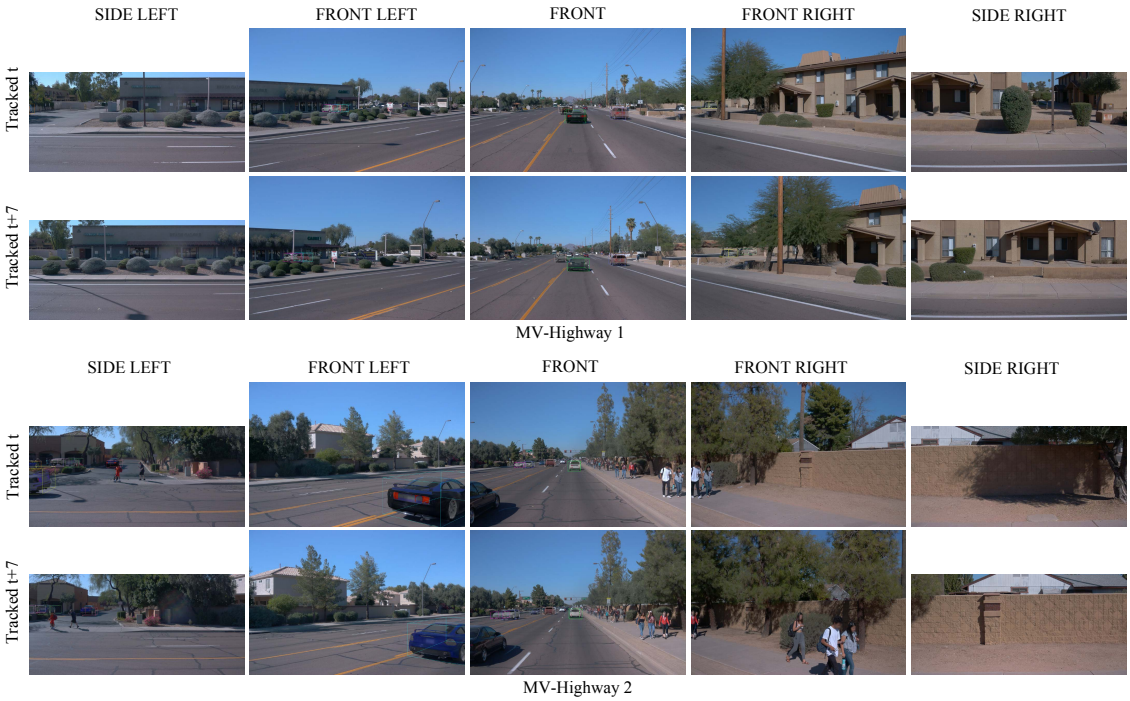


Supplementary Figure 4: Examples of failure cases, such as lighting (shadows and reflections) or occluded objects, where the reconstructed object differs significantly from the observed object. These visualizations allow us to understand exactly why our model fails in reconstructing and tracking objects. This also allows us to identify ways the representation model and perception pipeline can be improved to incorporate effects that cause the method to fail.

Supplementary Note 4: Computational Cost

Each inverse-rendered optimization step in our implementation takes ~ 0.3 seconds per frame. The overall computational cost of the method is determined by both the forward generation process and the gradient computation. However, we note that the rendering pipeline, contributing the majority of the computational cost of the generator, has not been performance-optimized and can be naïvely parallelized as follows:

- First, the loss and gradient computation can be parallelized in screen space within each camera view, enabling concurrent processing of multiple objects in all cameras rather than iterating over each camera individually. This results in a fraction $1/N_{objects}$ of the inference time in each tracking step for $N_{objects}$ objects that are tracked.
- Moreover, for multi-camera setups, the generation and gradient computation can be parallelized across all cameras. This further reduces the inference time to $1/N_{cameras}$ in each tracking step. Fig. 5 shows a typical multi-view setup from the Waymo Open Dataset.



Supplementary Figure 5: Multicamera Tracking on the Waymo Dataset [2]. The proposed directly supports joint tracking of objects across multiple cameras in typical autonomous vehicle setups [1,2]. All tracked objects and tracked bounding boxes are shown overlayed on the images. For all objects that are visible in more than one view, we only overlay the rendered representation on one image. For clarity and sparse observation in most camera views we naturally omit showing all cameras in the paper. Due to the 10Hz sampling rate of the Waymo Dataset, we show frames with a large timestep difference.

Supplementary Note 5: Differentiable Rendering Pipeline

A differentiable rendering method $R\left(o_p, c\right)$, such as rasterization for meshes or volumetric rendering for neural fields, renders an image $I_{c,p}$, a 2D observation of the 3D object o_p, given an object-centric camera projection $\mathbf{P}_c$. The projection is computed as $\mathbf{P}_c = \mathbf{K}_c\mathbf{T}$, where $\mathbf{K}_c$ is the camera intrinsic matrix and $\mathbf{T} = [\mathbf{R}|\mathbf{t}]$ a transformation to the camera c that is composed of rotation $\mathbf{R}$ and translation $\mathbf{t}$. While our method is general, implementation details of the generator and rendering method are provided in Supplementary Note 9.

Object poses are described by the homogeneous transformation matrix $\mathbf{T}_p \in \mathbb{R}^{4x4}$ with the translation $\mathbf{t}_p$ and orientation $\mathbf{R}_p$ in the reference coordinate system. The camera poses $\mathbf{T}_c \in \mathbb{R}^{4x4}$ is described in the same reference coordinate system. The relative transformation of the camera c and each object instance p can be computed through edge traversal in the scene graph as given in Eq. (2) of the main paper.

Supplementary Note 6: Detailed Optimization Objectives

This section provides detailed descriptions of the different components of our inverse rendering optimization objectives. Our objective consists of three components. Each of these is described in depth below.

The test-time optimized inverse rendering of all objects in every scene and across datasets minimizes the loss term

$$\begin{aligned}\mathcal{L}_{IR+embed} &= \mathcal{L}_{RGB} + \lambda \mathcal{L}_{perceptual} + \mathcal{L}_{embed} \\ &= \| \left(I_c - \hat{I}_c \right) \circ \hat{M}_{I_c} \|_2 \\ &+ \lambda_1 \text{LPIPS}_{patch} \left(I_c, \hat{I}_{c,p} \right) \\ &+ \lambda_2 \left(\alpha_S \mathbf{z}_S + (1 - \alpha_S) \mathbf{z}_S^{avg} \right) \\ &+ \lambda_3 \left(\alpha_T \mathbf{z}_T + (1 - \alpha_T) \mathbf{z}_T^{avg} \right)\end{aligned} \tag{1}$$

This combines RGB-MSE and a learned perceptual loss term with a regularization term, for which we describe implementation details as follows.

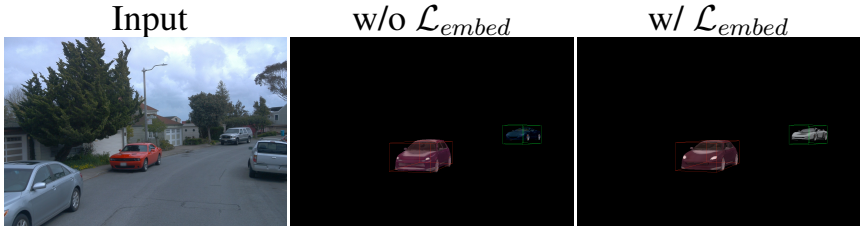
RGB Loss. We optimize only on rendered RGB pixels and minimize

$$\mathcal{L}_{RGB} = \| \left(I_c - \hat{I}_c \right) \circ \hat{M}_{I_c} \|_2, \text{ with } \hat{M}_{I_c} = \min \left(\sum M_{c,p}, \mathbf{1} \right). \tag{2}$$

The RGB loss is computed as the pixel-wise ℓ_2-norm. Only pixels inside a mask are considered for which each pixel in the composed image at least one of the objects in the respective frame c also projects too. Only pixel values inside this mask $\hat{M}_{I_c}$ are considered in the loss function. Please note that in scenes with occluded objects only the object closest to the camera contributes to the rendered pixel in this non-volumetric rendering pipeline. We also assume this for the input images, given that considered objects are solid and mostly non-transparent.

Learned Perceptual Loss. The mask of all foreground/object pixels $\hat{M}_{I_c}$ is computed as the sum over all object masks $M_{c,p}$ in the frame rendered by camera c. We employ a learned perceptual similarity metric [4] (LPIPS) on object-centered image patches, that is

$$\mathcal{L}_{perceptual} = \text{LPIPS}_{patch} \left(I_c, \hat{I}_{c,p} \right). \tag{3}$$



Supplementary Figure 6: Truncation Trick. From left to right, (i) The input observed image, (ii) results without truncation regularize applied and (iii) results with a truncation regularize applied. For images (ii) and (iii), we also show bounding boxes for each object color-coded by the respective predicted object IDs. As shown here, applying the truncation regularizer helps us achieve more accurate textures and better shapes and colors for the predicted car surface by forcing the optimized embedding to be "well behaved", i.e., close to the distribution of latent embeddings seen during the training by our representation model.

The goal of the perceptual loss is to guide the inverse rendering to match the abstracted appearance of the feature level of individual objects. We use the pre-trained LPIPS [4] loss with VGG16 [5] backbone for this, which operates on rectangular images with a minimum side length of 16 pixels. To consider objects individually we crop and resize patches from $\hat{I}_c$ for each object p respectively Mask $M_{c,p}(i,j)$ describes all pixels rendered from each object with their respective index i, j. $M_{c,p}(i,j) = 1$ if object p is projected into the respective pixel i, j from camera c. We then can describe an image patch by its top and bottom corner. The patch top corner is defined as $(u_{top}, v_{top}) = \left(i_{min,M_{c,p}}, j_{min,M_{c,p}}\right)$, the upper and left corner of a tight axis-aligned rectangle around all rendered pixels. The patch bottom corner is defined as $(u_{bot}, v_{bot}) = \left(i_{max,M_{c,p}}, j_{max,M_{c,p}}\right)$, the lower and right corner of the same axis-aligned rectangle.

Latent Embedding Regularization. The sum of Eq. 2 and 3 form the combined loss for the proposed differentiable rendering pipeline, which we optimize by refining the latent codes of shape and appearance, position, rotation, and scale.

$$\hat{\mathbf{z}}_{S,p}, \hat{\mathbf{z}}_{T,p}, \hat{s}_p\hat{\mathbf{t}}_p, \hat{\mathbf{R}}_p = \arg\min\left(\mathcal{L}_{IR}\right). \tag{4}$$

Modern generative-adversarial (GAN) [6–10] methods, such as the used 3D object generator [8] first map an embedding sample from a multivariate Gaussian, called **z**-space or distribution, into a learned embedding space, called **w**-space, following a different distribution. The intuition behind this is that there are more optimal embedding distributions, which can be more easily mapped to the data distribution that the GAN is generating. High-quality samples are only generated from embeddings inside the high-dimensional embedding distribution. Therefore, we regularize the optimized embedding code through inverse rendering, with

$$\mathcal{L}_{embed} = \|\alpha_T\mathbf{z}_T + (1-\alpha_S)\mathbf{z}_T^{avg}\| + \|\alpha_T\mathbf{z}_S + (1-\alpha_S)\mathbf{z}_S^{avg}\|, \tag{5}$$

that is Eq. (7) in the main paper. Here, $\mathbf{z}_S \in \mathbb{R}^{d_S}$ and $\mathbf{z}_T \in \mathbb{R}^{d_T}$ denote the latent vectors for shape and texture embeddings, respectively. The terms $\mathbf{z}_S^{avg}$ and $\mathbf{z}_T^{avg}$ are the mean embeddings of the prior model for shape and texture, while α_S and α_T control the weight of the regularization term for each embedding type. This loss function computes a weighted distance between each latent dimension of the individual embedding vectors and their respective mean embeddings. Mean embeddings $\mathbf{z}_S^{avg}$ and $\mathbf{z}_T^{avg}$ are obtained from sampled embeddings inside the distribution of the respective generative model. By summing across all dimensions and normalizing by the embedding size d_S and d_T, the resulting loss $\mathcal{L}_{embed}$ becomes scalar. Note that α_T and α_S are set to 0.7.

We employ the *truncation trick* widely used in GAN-based generators and first presented in StyleGAN [6], where z^{avg} represents an exponential-moving average of embedding codes generated from the Gaussian training during the training of the generator. This stabilizes the optimization through inverse rendering as Fig. 6 shows.

Instead of using vanilla stochastic gradient descent methods, we propose an alternating optimization schedule of distinct properties that includes aligning z_S before z_T, to reduce the number of total optimization steps. We first optimize a coarse color. Then, we jointly optimize the shape and positional state of each object. As the backbone of the learned perceptual loss, we use a pretrained VGG16 [5] and utilize individual output feature map similarities at different optimization steps. We find that color and other low-dimensional features are represented in the initial feature maps and those are better guidance for texture than high-dimensional features as outputs of the later blocks. These features have a more informative signal for shape and object pose. We use the average of the first and second blocks when optimizing for $\mathbf{z}_T$, while the combined perceptual similarity loss guides the optimization of $\mathbf{z}_T$ and the pose.

Weighting With empirical analysis of the validation set of both datasets, we find the weighting of loss terms $\lambda_1 = 0.4$, $\lambda_2 = 3$, and $\lambda_3 = 10$ for stable and truthfully generated objects via inverse rendering.

Supplementary Note 7: Detailed Optimization Schedule

We provide additional details on the optimization schedule used in our inverse rendering step. Supplementary Tab. 3 details the learning rates for each of the parameters in the loss function presented in Eq. 4 in Supplementary Note 6 in each of the 6 inverse rendering optimization steps of our method.

For the loss function presented in Eq. 1 in Supplementary Note 6, we found that the schedule in Tab. 3 solves this optimization problem effectively while being stable across various scenes and datasets.

We first fit the texture embedding in only three steps during the test-time optimization of all object parameters. In step 3, we jointly solve for pose, scale, and shape, followed by three more steps on only shape. Details on the learning rate for the respective parameters are reported in Tab. 3.

An exhaustive search is impossible due to the number of hyper-parameters when different loss functions and terms are included. We therefore performed empirical investigation on small, diverse subsets of scenes to find the parameter set used. The same setting works well on all datasets and *have not been changed* for the Waymo Open Dataset [2] and the nuScenes dataset [1].

Step \ Parameter (learning rate)	$\mathbf{z}_S$	$\mathbf{z}_T$	$\mathbf{t}$	Φ	s
1	-	3×10^{-1}	-	-	-
2	-	3×10^{-1}	-	-	-
3	6×10^{-2}	3×10^{-1}	3×10^{-2}	3×10^{-2}	1×10^{-6}
4	6×10^{-2}	-	-	-	-
5	6×10^{-2}	-	-	-	-
6	6×10^{-2}	-	-	-	-

Supplementary Table 3: Optimization Schedule. Test time optimization of all object parameters, the shape and texture embeddings $\mathbf{z}_S, \mathbf{z}_T$, their location $\mathbf{t}$, rotation Φ in $\mathfrak{so}(3)$ and scale s follows this schedule. First, the texture is fitted in for two steps, followed by a pose adjustment in one step and inverse rendering of the shape, defined by the respective embedding code. Green denotes the optimization of the parameter in the respective step. The learning rates for all optimized parameters are noted in each field.

Supplementary Note 8: Additional Tracking Details

In this section, we provide additional details about our inverse rendering method for 3D object tracking, including the prediction, interpretable latent matching, tracking State and embedding update steps, and tracking heuristics used. Algorithm 1 lists our entire end-to-end tracking method in pseudo-code format. For the predict, match and update step of the integrated Kalman Filter [11], we provide a detailed description and mathematical derivation below the algorithm in this section, see Eq. 6-12. In the algorithm, we introduce the variable $n_{lost,p}$ for each object p, that represents the count for the number of frames an object is lost, in addition to object states and appearance codes $\mathbf{z}_S$ and $\mathbf{z}_T$. If the counter reaches $N_{life} + 1$, then tracklets are not further tracked.

Algorithm 1 End-to-End Algorithm for Tracking via Inverse Rendering

Input: Initial 3D detections $\{\mathbf{x}_{det,k}\}$, frames $\{I_k\}$
Output: Active object states $\mathcal{X}$ across all frames $\{I_k\}$

Initialize $\mathcal{X} \leftarrow \emptyset$ ▷ List of tracked object states
Initialize Kalman filter [11] parameters $\mathbf{P}_{k-1}$, and $\mathbf{Q}$
for all $k \in [0, |\{I_k\}|]$ **do**
  if $|\mathcal{X}| > 0$ **then** ▷ **Step 1: Predict**
    $\hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_k \leftarrow \text{KALMANFILTERPREDICT}\big(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{P}_{k-1}, \mathbf{Q}\big)$ ▷ See Eq. 6
  end if

  Initialize $\mathbf{z}_S, \mathbf{z}_T, \mathbf{T}_{c,p} \; \forall \; \mathbf{x}_{det,k}$ ▷ **Step 2: Match**
  for all $n \in [0, N_{optim}]$ **do**
    $\hat{I}_c \leftarrow \text{DIFFERENTIABLERENDERER}(\mathbf{z}_S^{(n)}, \mathbf{z}_T^{(n)}, \mathbf{T}_{c,p}^{(n)})$ ▷ See Eq. 6 in the main paper
    $\mathcal{L}_{IR} \leftarrow I_c - \hat{I}_c$ ▷ Loss Computation, See Eq. 1
    $\mathbf{z}_{S,p}^{(n+1)} = \mathbf{z}_{S,p}^{(n)} - \omega_{z,S} \nabla_{\mathbf{z}_{S,p}} \mathcal{L}_{IR}$;
    $\mathbf{z}_{T,p}^{(n+1)} = \mathbf{z}_{T,p}^{(n)} - \omega_{z,T} \nabla_{\mathbf{z}_{T,p}} \mathcal{L}_{IR}$;
    $\mathbf{T}_{c,p}^{(n+1)} = \mathbf{T}_{c,p}^{(n)} - \omega_T \nabla_{\mathbf{T}_{c,p}} \mathcal{L}_{IR}$ ▷ Gradient Update, See Eq. 4
  end for

  $\hat{\mathbf{y}}_k = \mathbf{z}_{S,p}, \mathbf{z}_{T,p}, \mathbf{T}_{c,p}$
  $A \leftarrow \text{COMPUTEAFFINITY}\big(\hat{\mathbf{y}}_k, \hat{\mathbf{x}}_{k|k-1}\big)$ ▷ See Eq. 7
  $\mathcal{M}, \mathcal{U}_{tracked}, \mathcal{U}_{det} \leftarrow \text{HUNGARIANMATCH}(A)$ ▷ Matched and unmatched tracks and detects

  for all $m \in \mathcal{M}$ **do** ▷ **Step 3: Update**
    $\hat{\mathbf{x}}_{k|k}, \mathbf{P}_k, \mathbf{Q} \leftarrow \text{KALMANUPDATE}\big(\hat{\mathbf{y}}_k, \mathbf{P}_{k|k-1}, \mathbf{Q}\big)$ ▷ See Eq. 9, 10, 11
    $\mathbf{z}_{k,\text{EMA}} \leftarrow \beta \mathbf{z}_k + (1-\beta)\mathbf{z}_{k-1,\text{EMA}}$ ▷ See Eq. 8
    $n_{\text{lost, m}} \leftarrow 0$
  end for

  for all $u \in \mathcal{U}_{tracked}$ **do**
    $n_{\text{lost, u}} \leftarrow n_{\text{lost, u}} + 1$ ▷ Count unmatched time for tracklets
    if $n_{\text{lost, u}} > N_{\text{Life}}$ **then** ▷ Remove dead tracklets
      $\mathcal{U}_{tracked} \leftarrow \mathcal{U}_{tracked} \setminus \{u\}$
    end if
  end for

  $\mathcal{X} \leftarrow \mathcal{M} \cup \mathcal{U}_{tracked} \cup \mathcal{U}_{det}$ ▷ Add unmatched detections as new objects
end for

Prediction.

While not confined to a specific dynamics model, we use a linear State-transition model $\mathbf{A}$, For the objects kinematic state $\mathbf{x}_k = [x, y, z, s, \psi, w, h, l, x', y', z']_k$, and a forward prediction using a Kalman Filter [11], a vanilla approach in 3D object tracking [3]. An instantiated object in $k-1$ can be predicted in frame k as

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{A}\hat{\mathbf{x}}_{k-1|k-1} \\ \text{and } \mathbf{P}_{k|k-1} &= \mathbf{A}\mathbf{P}_{k-1|k-1}\mathbf{A}^T + \mathbf{Q}, \end{aligned} \tag{6}$$

with the predicted *a priori* covariance matrix modeling the uncertainty in the predicted State.

Interpretable Latent Matching.

In the matching stage, all optimal object representations o_p in frame k are matched with *tracked* and *lost* objects from $k-1$. Objects are matched based on appearance and kinematic State with a weighted affinity score

$$A = w_{IoU} A_{IoU,3D} + w_z A_z + w_c D_{centroid}, \tag{7}$$

where $A_{IoU,3D}$ is the IoU of the 3D boxes computed over the predictions of tracked object predictions $\mathbf{x}_{k|k-1}$ and refined observations. Here, the object affinity A_z is computed as the cosine distance of tracked object latent embeddings $\mathbf{z}$. In addition to that the Euclidean distance between the center $D_{centroid}$ adds additional guidance. We add no score For unreasonable distant tracked objects and detections.

We compute the best combination of tracked and detected objects using the Hungarian algorithm [12], again a conventional choice in existing tracking algorithms. Matched tracklet and object pairs are kept in the set of *tracked* objects and the corresponding detections representation is discarded. Unmatched detections are added as new objects. Unmatched tracklets are set to *lost* with a lost frame counter of one. Objects not detected in previous frames are set to *tracked* and their counter is reset to 0. Objects with a lost frame count higher than lifespan N_{life}, or outside of the visible field, are removed.

Tracking State and Embedding Update.

In the update step, we refine each object embedding $\mathbf{z}$ and expected observation $\mathbf{y}_k$ given the result of the matching step. Embeddings are updated through an exponential moving average

$$\mathbf{z}_{k,EMA} = \beta \mathbf{z}_k + (1-\beta)\mathbf{z}_{k-1,EMA} \text{ with } \beta = \frac{2}{T-1} \tag{8}$$

over all the past observations of the object. Here, T is the number of observed time steps of the respective instance. The observation $\mathbf{y}_k$ is used to update the Kalman filter. The optimal Kalman gain

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + R)^{-1} \tag{9}$$

is updated to minimize the residual error of the predicted model and the observation. The observation $\mathbf{y}_k$ is used to estimate the object State as

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(\mathbf{y}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}) \tag{10}$$

and with

$$\mathbf{P}_k = \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}\mathbf{P}_{k|k-1} \tag{11}$$

the *a posteriori* of the covariance matrix is updated.

Tracking Heuristics Above, we described the integration of test-time optimized object representations through inverse rendering into the tracking pipeline via an affinity score as defined in Eq. 7 as follows.

$$A = w_{IoU}A_{IoU} + w_z A_z + w_c D_{centroid},$$

that describes the similarity of tracked and detected objects in the matching step. We follow [1,3] and assign 0 to all object pairs whose center is more than $10m$ apart. In all other cases, we apply the weights $w_{IoU} = 0.7$, $w_z = 0.4$, and $w_c = 0.5$ between different affinity parameters. Here, A_{IoU} is the true 3D IoU of the bounding boxes, which is computed with the efficient PyTorch3D [13] implementation. The affinity A_z is computed as the pair-wise cosine distance between all tracklet-detection pairs. Centroid distances between pairs are computed in addition to the IoU to compensate for larger errors in line with the camera axis common in vision-based object detectors. In such cases, the IoU might be low, but object distances are still in a reasonable range which we empirically found at $5m$ For the object detector used. Finally, we define the distance-based affinity score as

$$D_{centroid} = \text{maximum}\left(\frac{-\|\mathbf{t}_{tracklet} - t_{detection}\|_2}{d_{max}}, 0\right) \text{, with } d_{max} = 5\text{m}. \tag{12}$$

Matches below the threshold of 0.48 For their affinity score are not considered matched and the respective tracklet is set to "lost" and the detection is added as a new tracklet in the next step. Tracklets are considered "dead" and removed after a maximum of 4 consecutive lost steps.

Supplementary Note 9: Generative Object Model

We provide further details of the generative object model used in our proposed method, including a description of the architecture, rendering process, and training details.

We employ the GET3D [8] architecture as object model G. Following StyleGAN [6,7] embeddings z_T and z_S are mapped to intermediate style embeddings $\mathbf{w}_S$ and $\mathbf{w}_T$ in a learned $\mathbf{W}$-space, which we optimize over instead of $\mathbf{Z}$-space. Style embeddings condition a generator function that produces tri-planes representing object shapes as Signed Distance Fields (SDFs) and textures as texture fields. We deliberately *train our generator on synthetic data only*, see experiments below. Differentiable marching tetrahedra previously introduced in DMTet [14] extract a mesh representation and Images are rendered with a differentiable rasterizer [15].

The object generator, used as the prior for car representation, follows the architecture from GET3D [8]. The generator maps two sample codes z_{tex} and z_{geo}, drawn from a Gaussian distribution for the texture and shape respectively, to samples of a 3D SDF and a texture field. Details of this object model's architecture, training, and usage are below.

Generator Architecture. In this work, we employ a variant described in the appendix of [8]. Following [6,7], two Gaussian variables are mapped independently to intermediate style embedding $\boldsymbol{w}_S$ and $\boldsymbol{w}_T$ in a learned $\boldsymbol{W}$-space. Style embeddings are then used as an input to the CNN-based StyleGAN2 [7] generator. Both style embeddings for shape and texture jointly condition the generator in each block, allowing texture and shape to influence the other modality. Each intermediate feature map of the generator backbone is mapped to its respective modality through a mapping layer only conditioned on the respective style embedding. All feature maps are accumulated and reshaped into three feature planes in the last generator layer. These planes represent textures as texture fields, object shapes as Signed Distance Fields (SDFs), and vertex offsets of a corresponding mesh. This forms a feature volume representation of the textured 3D object on two tri-planes.

Rendering. We employ the differentiable marching tetrahedra [14] method and extract a mesh representation from the SDF and vertex offsets, allowing more efficient rendering when compared to sampling-based volumetric SDF rendering. Differentiable rasterization for meshes efficiently renders a 2D image of the respective mesh. The respective vertex color can be efficiently retrieved to render the RGB image output by querying the texture field only at visible surface points.

Training. The model is trained using adversarial losses defined on the 2D renderings of 3D objects from the ShapeNet version 1 dataset [16]. Specifically, we use a differentiable rasterizer

to render RGB images together with the silhouette masks of the objects as in [8] with a training configuration that largely follows StyleGAN2 [7] including using a mini-batch standard deviation in the discriminator, exponential moving average for the generator, non-saturating logistic loss, and R1 regularization.

Supplementary Note 10: Fair Comparison to End-to-end Approaches

In this section, we provide a fair comparison of our method's performance to tracking methods that are trained end-to-end. Supplementary Figure 7 compares 3D bounding box outputs from QD-3DT [17] trained on Waymo [2] and our inverse neural rendering based tracker (INR) overlaid on the respective input videos from nuScenes [1].

The naive quantitative evaluation of multi-object tracking methods can easily be "unfair" in the sense that the tracker during training may be given access to future frames or rely on an improved detector backbone (making it challenging to evaluate the tracker in isolation). Evaluating generalization requires a nuanced setup to provide a fair evaluation. Therefore, we decide to focus the evaluation on methods that either build on the same detector backbone [18] and are not trained on the respective training set [3], achieving generalization by design, or end-to-end trained tracking methods for which we use a model trained on a different dataset. In particular, we evaluate a model of QD-3DT [17] that has been trained on the Waymo Open Dataset [2] on nuScenes [1]. The authors of QD-3DT [17] were so kind to provide the respective checkpoints to us.



Supplementary Figure 7: Qualitative Comparison on Generalization. We compare 3D bounding box outputs from QD-3DT [17] end-to-end trained on Waymo Open Dataset [2] and our inverse neural rendering-based tracker (INR) overlaid on the respective input videos from nuScenes [1]. The same color in consecutive frames denotes the same tracklet. As we see in scene (a) on the left side QD-3DT has ID switches especially in the far range and on the sides of a frame implying dataset-specific performance caps, e.g. the training dataset has been tracking annotated in a rather short range. Another finding is exemplified in scene (b) on the right side where the tracker does not generalize well and is losing a tracklet.

Fig. 7 shows tracking results from our method along with results from QD-3DT. A visual, qualitative inspection illustrates that the end-to-end trained tracking method QD-3DT [2] still performs well on objects in the center of the scene but does not generalize well on occluded or partially visible objects. This is reflected in the scores reported in Fig. **2 a** of the main manuscript.

Supplementary Note 11: Multi-Class and Human Generator Extension

We provide additional experiments that confirm the generalization of our method to object classes other than vehicles and use several different generator models. To this end, we extend our method and analyze results on the `motorcycle` class using a GET3D model variant [8] trained on synthetic motorcycles and the `pedestrian` class by integrating the human avatar model EVA3D [19] as a generative prior for modeling pedestrians with realistic shapes and poses.

Car and Motorcycle Multi-Class Extension and Evaluation. To accommodate for different classes, we apply the same differentiable rendering function and same tracking method as described in the main manuscript. Only two adaptions are made to accommodate for tracking N_{class} classes. First, the different prior distribution model is selected according to the detected or tracked class label. Additionally, we only perform matching and calculate affinity between observations $\mathbf{y}_k$ and tracked objects $\mathbf{x}_k$ of the same class label, resulting in N_{class} separate affinity score calculations. No further changes to the method were made, verifying extendability beyond single-class tracking with the proposed differentiable rendering pipeline. Qualitative and quantitative results on the `motorcycle` class are presented in the table of main paper Figure 1 and in Supplementary Figure 8 that confirm tracking at a quality similar to the vehicle class with respect to baseline approaches, indicating robustness across rigid object classes.

Deformable Object Classes. The `pedestrian` class introduces unique challenges, including higher pose variability, non-rigid deformations, and frequent occlusions in crowded scenes, and, as such, requires a non-trivial extension to the proposed method. Nevertheless, we find that the method is capable of supporting rigid approximations of `pedestrians`, by integrating the respective generative prior model Eva3D [19]. Integration of this class follows the process described above. In addition to that, we only optimize the respective texture representation $\mathbf{z}_T$ on top of the gender-neutral SMPL body model [20] and omit the optimization of shape representation $\mathbf{z}_S$, that is explicitly modeled as deformation parameters of the body model.

We confirm in our experiments in Supplementary Figure 8 interpretable tracking with good performance with the rigid approximation, validating the adaptability and versatility of our method to diverse tracking scenarios. Nevertheless, deformability of body and cloth, estimating gender, and unique clothing choices remain challenges. While outside the scope of our work, the optimization of deformable object parameters, such as skeleton tracking [20], through inverse rendering presents an interesting avenue for future work.



Supplementary Figure 8: Multi-Class Tracking with Pedestrians and Motorcycles. We show three distinct scenes from the NuScenes [1] dataset on multi-class tracking, combining the Get3D [8] vehicle, motorcycle and Eva3D [19] pedestrian model in a single differentiable rendering pipeline. The input image of the first frame of the sequence is shown on the left, while the remaining frames provide images from sequential timesteps overlayed with the retrieved rendered objects. Even in more crowded scenes (top) with multiple object classes and inter-object occlusion, all objects can be consistently tracked across timestep. Further observations of occluded and less crowded scenes show consistent appearances and tracking of pedestrians and a motorcycle.

Supplementary Note 12: Practical Impact of Interpretable INR-based Tracking

This section expands on the practical applications of our approach, highlighting its utility and broader impact beyond multi-object tracking improvements. Allowing for interpretability, cost-effective dataset annotation, and extensibility to multi-camera and multi-sensor settings, the proposed method may enable several applications in autonomous driving and robotics beyond 3D object tracking and scene understanding. We describe these application areas in detail in the following.

Generalizable Offline Auto-Annotation. Being data-agnostic, the method has high potential offline auto-annotation, addressing a significant challenge in creating large-scale annotated datasets for 3D object tracking for autonomous driving [9, 10], while maintaining human interpretable reconstructions for QA processes and providing generalizability. As such, the accurate and interpretable 3D reconstructions produced by our method may reduce the cost of generating annotated datasets for 3D object tracking, a critical bottleneck in autonomous driving research.

Interpretability for Debugging Perception Pipelines. Our approach facilitates retrospective reconstruction and analysis of perception failure cases in real-world scenarios, providing feedback for debugging. For instance, in scenarios where the association between tracked object instances in adjacent frames fails, a visual model of all reconstructed objects allows for *debugging by inspection*. We show specific examples in Supplementary Note 3.

Potential for Multi-Camera or Multi-Modal Extensions. Our method is inherently suitable for extension to multi-camera see Fig. 5, or multi-modal setups. In contrast to feature-based models, our approach can naively integrate data (explaining multiple observations with additional loss components) from multiple overlapping viewpoints and modalities without incorporating complex and sensor-specific sensor fusion approaches by optimizing 3D object representations across multiple sensors. Although multi-modal setup is beyond the scope of this work, this extension holds promise for improving performance in scenarios with complex occlusions or sparse observations.

Supplementary Note 13: Validation of Retrieved Object Properties and Downstream Applications

Our inverse rendering-based tracking method estimates not only object trajectories but also retrieves additional object properties such as pose, shape, texture, and appearance. These properties offer richer scene understanding beyond traditional bounding-box-based tracking. In this section, we quantitatively evaluate the accuracy of these retrieved attributes and illustrate their use for downstream tasks such as free-space detection, motion planning, and failure case analysis.

Occupancy Estimation for Motion Planning Since our method reconstructs 3D object shapes, we can infer vehicle occupancy in a scene. The rendered geometry can be used as coarse occupancy predictions, aiding occupancy-based motion planning algorithms [21–23].

Instance-Level Segmentation The retrieved object (shape and pose) allows us to generate instance-level segmentation masks, as shown in Supplementary Figure 9. These masks can be directly leveraged for instance segmentation in image space [24]. We note that the method provides object segmentation here without requiring explicit segmentation labels to train a separate segmentation model.



Supplementary Figure 9: Rendered Instance Segmentation. Coarse instance segmentation can be directly obtained from the rendering pipeline for free. For each fitted object, we additionally render a depth-ordered alpha mask. This can be used to assign a per-pixel instance segmentation masks as shown on the right.

Image Reconstruction Quality To assess the accuracy of our retrieved object properties, we evaluate their quality using image-based metrics. Specifically, we measure how well the optimized

object parameters reconstruct the observed image regions using LPIPS [4] and PSNR. We report our in Table 4, and we find that our inverse-rendered reconstructions closely resemble the original observations, demonstrating the accuracy of our retrieved object attributes.

Supplementary Table 4: Image Reconstruction Accuracy Metrics. Values are averaged over objects from 20 scenes. We measure the respective reconstruction accuracy metric between the input RGB observation and the reconstructed rendered objects (after optimization), and the quality of the initial rendered object from initial latents (before optimization).

Reconstruction Metric	Before Optimization	After Optimization
PSNR ↑	12.9	15.1
1 - LPIPS ↓	0.87	0.86

References

1. Caesar, H. *et al.* nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11621–11631 (2020).

2. Sun, P. *et al.* Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2446–2454 (2020).

3. Weng, X., Wang, J., Held, D. & Kitani, K. 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10359–10366 (IEEE, 2020).

4. Zhang, R., Isola, P., Efros, A. A., Shechtman, E. & Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 586–595 (2018).

5. Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition (2015).

6. Karras, T., Laine, S. & Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4401–4410 (2019).

7. Karras, T. *et al.* Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8110–8119 (2020).

8. Gao, J. *et al.* Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, vol. 35, 31841–31854 (2022).

9. Sauer, A., Schwarz, K. & Geiger, A. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 conference proceedings*, 1–10 (2022).

10. Kang, M. *et al.* Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10124–10134 (2023).

11. Kalman, R. E. A new approach to linear filtering and prediction problems (1960).

12. Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly* **2**, 83–97 (1955).

13. Ravi, N. *et al.* Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501* (2020).

14. Shen, T., Gao, J., Yin, K., Liu, M.-Y. & Fidler, S. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems* **34**, 6087–6101 (2021).

15. Laine, S. *et al.* Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)* **39**, 1–14 (2020).

16. Chang, A. X. *et al.* ShapeNet: An Information-Rich 3D Model Repository. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015).

17. Hu, H.-N. *et al.* Monocular quasi-dense 3d object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

18. Zhou, X., Koltun, V. & Krähenbühl, P. Tracking objects as points. In *European Conference on Computer Vision (ECCV)*, 474–490 (Springer, 2020).

19. Hong, F., Chen, Z., LAN, Y., Pan, L. & Liu, Z. EVA3d: Compositional 3d human generation from 2d image collections. In *International Conference on Learning Representations* (2023). URL https://openreview.net/forum?id=g7U9jD_2CUr.

20. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G. & Black, M. J. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, 851–866 (2023).

21. Tong, W. *et al.* Scene as occupancy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 8406–8415 (2023).

22. Tsardoulias, E. G., Iliakopoulou, A., Kargakos, A. & Petrou, L. A review of global path planning methods for occupancy grid maps regardless of obstacle density. *Journal of intelligent & robotic systems* **84**, 829–858 (2016).

23. Guo, K., Liu, W. & Pan, J. End-to-end trajectory distribution prediction based on occupancy grid maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2242–2251 (2022).

24. Zhang, H. *et al.* A simple framework for open-vocabulary segmentation and detection. *arXiv preprint arXiv:2303.08131* (2023).